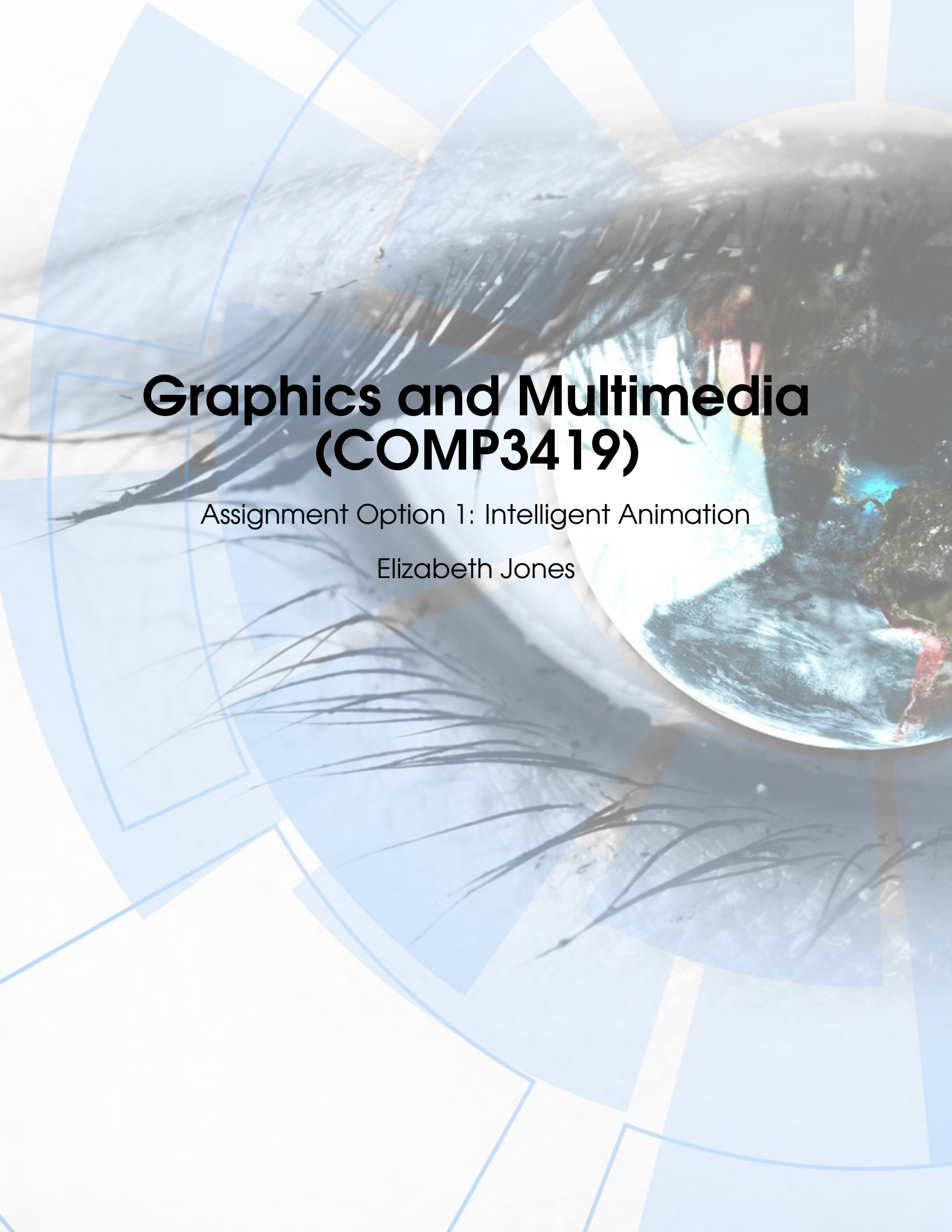
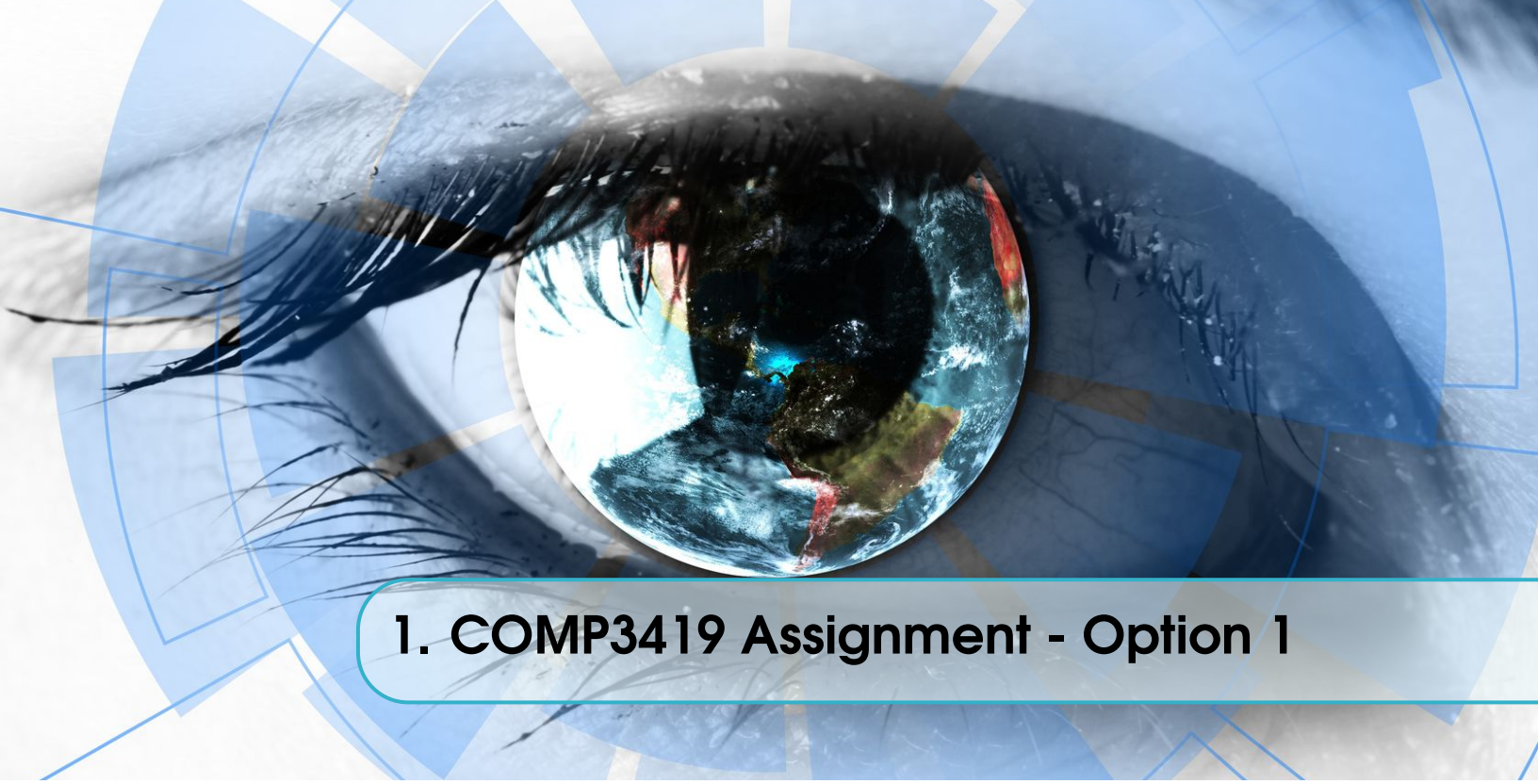


Graphics and Multimedia (COMP3419)

Assignment Option 1: Intelligent Animation

Elizabeth Jones



1. COMP3419 Assignment - Option 1

1.1 Introduction

I drew inspiration from the apple-catching arcade game that I used to play when I was little for this project.

Scene replacement

The video's blue backdrop transforms into two distinct kitchen settings, each representing a unique game level. The scenes transition to either red or green based on the player's success.

Main character

The monkey from the video morphs into a chef, mirroring the monkey's movements with synchronized hands, legs, and body.

Side characters

- The cookies: Cookies fall from the chef's hands, their movements controlled by the computer. There's always one cookie on the screen.
- The cookie jar: Representing the player, the user moves the jar to collect cookies. Successful catches increase the score while hitting the chef bounces the jar to the bottom left corner. Players can move the jar using arrow keys for up, down, left, and right movements.

Storyline

In level 1 and level 2, the score is displayed in the top right corner and the timer is displayed in the top left corner (counts up in seconds)

- Start Screen: The background is the basic kitchen with text at the bottom of the screen displaying "Ready to cook? (enter to start)"
- Level 1: After clicking "enter" from the start screen, the user enters the first level with a basic kitchen background. A timer starts, and the score is initialized to 0. Collecting three cookies before 30 seconds brings the user to the next level *(Event 1)*. Failure to do so results in the losing screen *(Event 2.1)*
- Level 2 *(Event 1)*: Users who passed Level 1 are brought to Level 2, featuring a new more professional-looking kitchen background. The timer and score reset and players now have 30 seconds to collect five cookies. Success leads to the winning screen *(Event 2.2)*, otherwise,

users are brought to the losing screen *(Event 2.1)*

- Won or Lost *(Event 2)*: For successful players, a green screen with the text “You won!” appears. Otherwise, a red screen appears with the text “You lost :(” appears. Both cases represent the end of the game.

1.2 Replace the Marionette

1.2.1 Extracting Frames

I used cv2 to extract each frame from the "Opt1-MarionetteMovements" video and stored each one in a "frames" folder. I saved the frame number, frame height, and frame width into their own variables to use later.

1.2.2 Applying Masks

Each frame I transformed into an HSV image, then followed these steps to filter out everything besides the red parts of the monkey (the center, hands, and feet).

1. Applied a red filter *(Figure 1.1a)* - I created a lower bound *rgb(0, 50, 150)*, and upper bound *rgb(20, 255, 255)* to best filter out everything in the frame beside the main, red points from the monkey. Then using cv2.inRange I applied the mask and converted the new image, with my mask, to black and white.
2. Noise filtered
   (a) dilated image *(Figure 1.1b)* - Since my red mask kept some of the monkey’s face I dilated the image to totally remove the outline of the face. I dilated it 3 times, using a 3x3 kernel of 1s.
   (b) eroded image *(Figure 1.1c)* - After dilating the points were disconnected, making more than 5 groups, and very small, so to make them more noticeable I eroded them. I used a 6x6 kernel of 1s and eroded it twice.

For each step, I used trial and error in my ranges, iterations, and kernel sizes until I was satisfied with my results and I saw 5 of the cleanest, largest groupings.

(a) Red Mask (b) Dilated Image (c) Eroded Image

Figure 1.1: Each step of masking and denoising a frame

1.2.3 Finding and Sorting Centroids

After researching cv2 methods, I came across centroids, on the website *geeksforgeeks* . I saw that it can find centroids from shapes, which I have from my previous step. I found all of my contours from my final mask and put them through my helper function "labelBodyParts" with the number of main points I wanted, 5. Then in my helper function, I sorted through the centroids using PriorityQueue(). I created a variable for the body and left and right hands and feet. I first found the center of the

marionette by going through each point in the given list and finding the maximum and minimum x and y values, then using them to find the approximated center. I then loop through each centroid and sort them based on how close they are to my approximated center. The first element of my PriorityQueue is my center. I then use that to find the left and right arm by adding centroids to a list only if they are above my body, and if the x-value is smaller than my body it is added to my leftArm queue, otherwise it is added to my rightArm queue. I then repeat that process with my feet and save them into variables leftFoot and rightFoot respectively. Since PriorityQueues sort negative numbers by their largest to smallest absolute values, for my right hand and foot I subtracted the body x-value by the potential point to get the furthest point to be the highest priority. Otherwise, as you can see in Figure 1.1c, another grouping would have wrongly been identified as the left hand. I then create an array of length k, 5, and place the leftHand, rightHand, body, leftFoot, and rightFoot. Then with this new list, back in my main method "findCentorids()", I iterate through each frame and get the centroid points, I then check that all five centroid points exist, and if they do I append them to my final list of centroids.

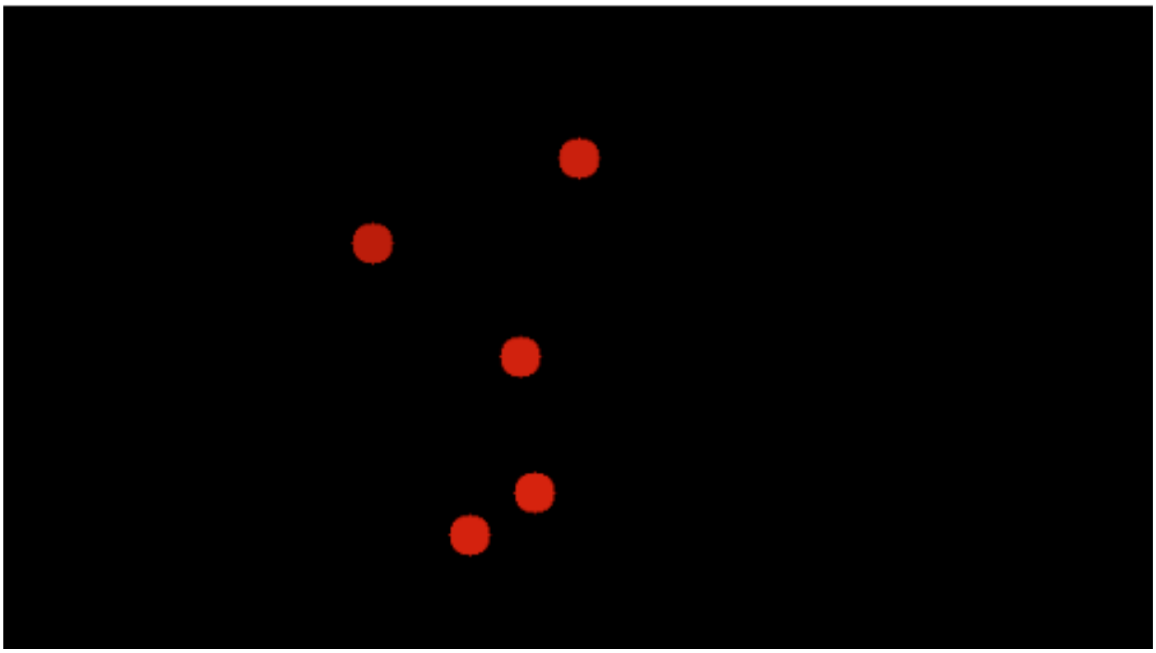


Figure 1.2: Final centroid points

1.3 Create the Pygame

1.3.1 Setting up my Game

After having my 2D list of centroids, I was able to start my game. I decided to use pygame, which made replacing the background of the video easy. I used two distinct kitchen backgrounds, and a green/red screen to replace the blue in the video depending on what stage the player was on. I took the height and width of my "frame0" to size my backgrounds. After that, I was able to draw everything else on top of those backgrounds. Then I created classes that I will need:

- GameRules - puts all my characters together

- CookieJar - the player, controls movement
- BodyPart - the centroid points
- Limbs - draws the arms and legs
- Cookie - falls from a given point
- CookingGame - main class

1.3.2 Adding Characters

Uploaded their images into respective variables during setup.

1. **Chef**
   The chef uses the centroids to mimic the monkeys movements. To draw the centroids and swicth between them at a speed of 30 frames per second, I used the pyGameClock, an index for where I am on my centroid list, and created a centroid switch time variable, 33 *(TotalTime ÷ NumberofSwitches)*, then while either level 1 or level 2 is running I check if the time since the last switch is greater than or equal to the centroid switch time, and if it is then I reset the time since last switch and update my index. I also then add the body parts from the index that I am currently on. Then I draw the arms and legs using the limbs class, which draws lines from the hands/feet to the body centroid.
2. **Cookie**
   The cookie falls from either the left or right hand at random, and there is always one cookie one the screen at a time. The x and y coordinates of the hand is sent into my cookie class and the cookie then falls from that point at a random speed.
3. **CookieJar**
   The player moves up, down, left and right with the arrow keys.

1.3.3 Making the Rules

Interactions between characters

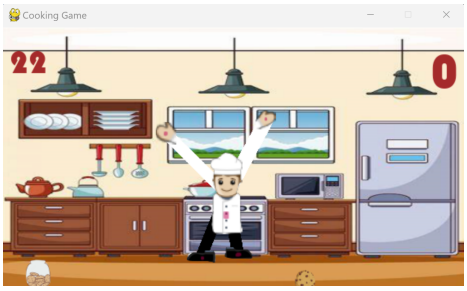
- If the user hits the chef, then it is bounced to the left corner of the game
- If the user catches a cookie then the score is increased
  - If the cookie is caught its coordinates get pushed way off the screen, so the score is only increased by the first interaction

The world is always generated with a CookieJar and a cookie. The game ends if the score is greater than the given max score, or if 30 seconds pass. Each tick, updates all the hands, feet, body, arms and legs, the cookie, the player, and the score, and checks if the cookie was caught or if the chef was hit.

1.3.4 Putting it all Together

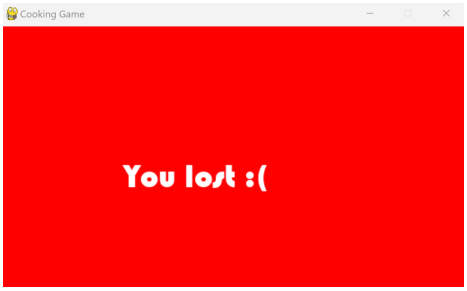
My main class, CookingGame, has four while loops that check different variables to determine what stage should be shown.

1. If the game has not started and the game is not over: Start screen (Figure 1.3e)
2. If the game has started and it is level one and the game has not been won: Level 1 (Figure 1.3a)
3. If the game has started and it is not level one and level two has not been won: Level 2 (Figure 1.3b)
4. If both games are over and both were won: Succeeded screen (Figure 1.3d)
5. If anything has been lost: Failed screen (Figure 1.3c)

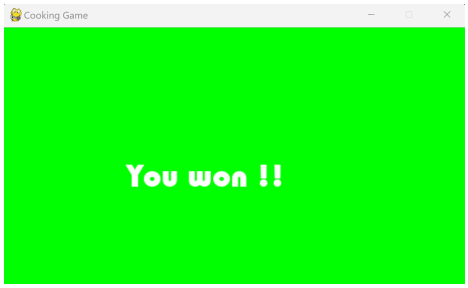


(a) Level 1

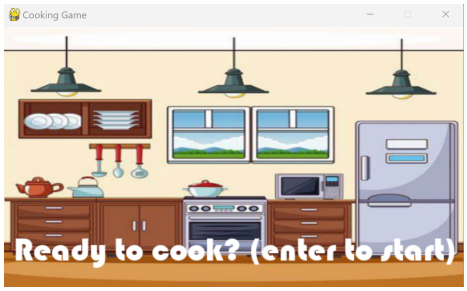
(b) Level 2



(c) Failed Screen



(d) Succeeded Screen



(e) Start Screen

Figure 1.3: Different stages of my game